A FAST ALGORITHM FOR PARALLEL COMPUTATION OF MULTIBODY DYNAMICS ON MIMD PARALLEL ARCHITECTURES

Amir Fijany*, Gregory Kwan*, and Nader Bagherzadeh†

*Jet Propulsion Laboratory, California institute of Technology

†Department of Electrical and Computer Engineering
University of California, Irvine

Abstract

in this paper the implementation of a parallel $O(Log N)$ algorithm for computation of rigid multibody dynamics on a Hypercube MIMD parallel architecture is presented. To our knowledge, this is the first algorithm that achieves the time lower bound of $O(Log N)$ by using an optimal number of O(N) processors. However, in addition to its theoretical significance, the algorithm is also highly efficient for practical implementation on commercially available MIMD parallel architectures due to its highly coarse grain size and simple communication and synchronization requirements. We present a multilevel parallel computation strategy for implementation of the algorithm on a Hypercube. This strategy allows the exploitation of parallelism at several computational levels as well as maximum overlapping of computation and communication to increase the performance of parallel computation.

J. Introduction

The multibody dynamics problem concerns the determination of the motion of the mechanical system resulting from the application of a set of control forces. In the context of robotic applications, the problem is more known as forward dynamics problem. In this paper we consider a multibody system with a serial chain topology. However, our results can be extended to systems with other topologies, e.g., closed-chain topology [1,2].

In brief mathematical terms, the multibody dynamics problem can be stated as the solution of a linear system

$$\bullet \quad Q = \mathcal{T} - b(\theta, \dot{Q}, F_E) = \mathcal{F}_T, \text{ or}$$

$$\ddot{Q} = \mathcal{M}^{-1}\mathcal{F}_T \qquad (1)$$

The vector $b(\theta, \dot{Q}, F_E)$ represents the nonlinear terms which can be computed by using the recursive Newton-Euler (N-E) algorithm [3] while setting the vector of joint accelerations, $\ddot{Q}$, to zero. In Eq. (1), $\mathcal{F}_T = \mathcal{T} - b(\theta, \dot{Q}) \triangleq \text{col}\{F_{Ti}\} \varepsilon \Re^N$ represents the acceleration-dependent component of the control forces.

At present it seems that the development of serial algorithms for the problem has reached a certain level of maturity. These algorithms can be classified as the $O(N^3)$ algorithm [4], the $O(N^2)$ algorithm [5], and the O(N) algorithms [6-9]. However, despite the significant improvement in efficiency of serial algorithms, even the fastest algorithm is still far from providing the real-time or faster-than-real-time simulation capability. This suggests that any further significant improvement in computational efficiency can only be achieved through exploitation of parallelism.

The development of efficient parallel algorithms for multibody dynamics is a rather challenging problem. It represents a very interesting example for which the analysis of the efficiency of a given algorithm for parallel computation is far different and more complex than that for serial computation. In fact, our previous analysis [10,1 1], which is also supported by the results of this paper, clearly indicate that those algorithms that are less efficient (in terms of either asymptotic complexity or number of operations) for serial computation provide a higher degree of parallelism and hence are more efficient for parallel computation.

A preliminary investigation of parallelism in computation of the problem by analyzing the efficiency of existing algorithms for parallel computation is reported in [10]. The main result of this investigation can be summarized as follows.

1. The existing O(N) algorithms are strictly serial, that is, their parallelization results in O(N) parallel algorithms which are faster than their serial

Nomenclature

N	Number of Degrees-Of-Freedom (DOF) of the system
m, a	Cost of multiplication and addition
$p_{i,j}$	Position vector from O_j to O_i, $p_{i+1,i} = p_i$
$c(i, i+1)$	3 x 3 matrix describing the orientation of frame i+1 with respect to frame i
m_i	Mass of link i
J,	Second moment of mass of link i about its center of mass
h_i, k_i	First and Second Moment of mass of link i about point O,
$\mathcal{M} \varepsilon \Re^{N \times N}$	Symmetric Positive Definite (SPD) mass matrix
θ_i, Q_i, $\dot{Q}_i$	Position, velocity, and acceleration of joint i
T,	Applied (control) force on joint i
$\omega_i, \dot{\omega}_i \varepsilon \Re^3$	Angular velocity and acceleration of link i
v_i, $\dot{v}_i \varepsilon \Re^3$	Linear velocity and acceleration of link i (point O_i)
f_i, $n_i \varepsilon \Re^3$	Force and moment of interaction between link i-1 and link i

Spatial Quantities

H_i	Spatial axis (map matrix) of joint i, $H_i \varepsilon \Re^{6 \times k}$ for a joint with k DOFs
$C(i, i+1) \varepsilon \Re^{6 \times 6}$	A 6 x 6 matrix used for projection of spatial vectors, given in frame i +1, onto frame i

$$C(i, i+1) = \begin{bmatrix} c(i,i+1) & 0 \\ 0 & c(i,i+1) \end{bmatrix}$$

$I_{i,j} \varepsilon \Re^{6 \times 6}$	Spatial Inertia of body i about point O_j, $I_{i,i} = I_i$

$$I_i = \begin{bmatrix} k_i & \tilde{h}_i \\ \tilde{h}_i^T & m_i U \end{bmatrix}$$

(~ denotes transpose)

$V_i = \begin{bmatrix} \dot{\omega}_i \\ \dot{v}_i \end{bmatrix} \varepsilon \Re^6$	Spatial acceleration of link i
$F_i = \begin{bmatrix} n_i \\ f_i \end{bmatrix} \varepsilon \Re^6$	Spatial force of interaction between link i-1 and link i
$F_E \varepsilon \Re^6$	External spatial force acting on the End-Effector (EE)

Global Quantities

$\mathcal{H} \triangleq \text{diag}\{H_i\}$	Global matrix of spatial axes, $\mathcal{H} \varepsilon \Re^{6N \times N}$ for a system with 1 DOF joints.
$\mathcal{I} \triangleq \text{diag}\{I_i\} \varepsilon \Re^{6N \times 6N}$	Global matrix of spatial inertia, i = N to 1
$\Theta \triangleq \text{col}\{\theta_i\} \varepsilon \Re^N$	Global Vector of joint positions, i = N to 1
$\mathcal{Q} \triangleq \text{col}\{Q_i\} \varepsilon \Re^N$	Global vector of joint velocities, i = N to 1
$\dot{\mathcal{Q}} \triangleq \text{col}\{\dot{Q}_i\} \varepsilon \Re^N$	Global vector of joint accelerations, i = N to 1
$T \triangleq \text{col}\{\tau_i\} \varepsilon \Re^N$	Global vector of applied joint forces, i = N to 1
$\dot{\mathcal{V}} \triangleq \text{col}\{\dot{V}_i\} \varepsilon \Re^{6N}$	Global vector of spatial accelerations, i = N to 1
$X \triangleq \text{col}\{F_i\} \varepsilon \Re^{6N}$	Global vector of spatial interaction forces, i = N to 1

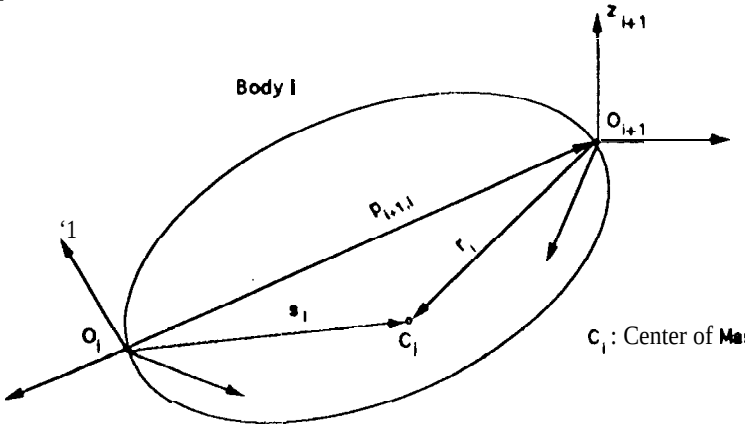

Figure 1. Body, Frames, and Position Vectors

counterparts only by a small constant factor.

2. Theoretically, the time lower hound of $O(Log^2 N)$ can be achieved by parallel ization of the $0(N^3)$ algorithms by using $O(N^3)$ processors.

3. Practically, the best parallel algorithm results from parallelization of the $O(N^3)$ algorithm by using a two-dimensional array of $O(N^2)$ processors.

1 he analysis in [10] also led to two important conclusions The first was that, if indeed there can be a both time and processor-optimal parallel algorithm for the problem, i.e., a parallel algorithm achieving the time lower bound of $O(Log N)$ with O(N) processors, then this parallel algorithm can only he derived by parallelization of an O(N) serial algorithm, Since the existing O(N) algorithms are strictly serial, the second conclusion was that the first step towards developing such an optimal parallel algorithm is to devise new O(N) algorithms with efficiency for parallel computation in mind. Such algorithms can only be derived by a global reformulation of the problem and not an algebraic transformation in t he computat ion of existing O(N) algorithms.

Physically, a given algorithm for multibody dynamics can be classified based on its force decomposition strategy. Mathematically, the algorithm can he classified based on the resulting factorization of mass matrix which corresponds to the specific force decomposition (see [1 1,12] for a more detailed discussion). A new algorithm based on a global reformulation of the problem is then the one that starts with a different and new force decomposition strategy and results in a new factorization of mass matrix.

Interestingly, a recently developed iterative algorithm in [13, 14] represents such a global reformulation of the problem. It differs from the existing O(N) algorithms in the sense that it is based on a different strategy for force decomposition. We have shown that this strategy leads to a new and completely different factorization of *M-l* [1 1,12]. 'l'his factorization, in turn, results in a new O(N) algorithm for the problem which is designated as the ***Constraint Force*** (CF) algorithm. A salient feature of the CF algorithm is that it is ***strictly efficient for*** *parallel* ***computation***, that is, it is less efficient than other O(N) algorithms for serial computation but it can be fully parallelized leading to a both time and processor-optimal parallel algorithm for the problem, i.e., a parallel $O(Log N)$ algorithm with O(N) processors.

This parallel algorithm- in addition to being theoretically significant by proving, for the first time, the existence of a both time and processor-optimal parallel algorithm for the pro blem- is also highly practical from an implementation point of view. This is due to its large grain size and simple communication and synchronization requirements. This paper is organized as follows. in §II, we briefly review the CF algorithm. Serial implementation of the CF algorithm is discussed in §III. The multilevel parallel computation of the CF algorithm on the Hypercube is presented in §IV. Finally, some discussion and concluding remarks arc made in §V.

II. The Constraint Force Algorithm

A. Notation and Preliminaries

In the following derivation, we make usc of spatial notation (shown with upper-case *MATHSITALIC* letters) and global notation (shown with upper-case *CALLIGRAPHIC* letters) which lead to a compact representation of equations. Here, only joints with one revolute DOF are considered. However, the results can he extended to systems with joints having different arid/or more DOFs.

Wit h any vector v, a tensor $\tilde{v}$ can be associated whose representation in any frame is a skew symmetric matrix:

$$\tilde{v} = \begin{matrix} 0 \\ v_{(z)} \\ [-v_{(y)} \end{matrix} \quad \begin{matrix} -v_{(z)} \\ 0 \\ v_{(x)} \end{matrix} \quad \begin{matrix} 1'(Y) \\ -v_{(x)} \\ 0 \quad 1 \end{matrix}$$

where $v_{(x)}, v_{(y)}$, and $v_{(z)}$ arc the components of v in the frame considered. The tensor $\tilde{v}$ has tile properties that $\tilde{v}^T = -\tilde{v}$ and $\tilde{v}_1 v_2 = v_1 \times v_2$, i.e., it is a vector cross-product operator. The matrix V associated with the vector u is defined as

$$V = \begin{bmatrix} U & \tilde{v} \\ 0 & U \end{bmatrix} \quad \text{and} \quad V^T = \begin{bmatrix} U & 0 \\ -\tilde{v} & U \end{bmatrix} \varepsilon \Re^{6x6}$$

where here (as well as through the rest of the paper) U and O stand for identify and zero matrices of appropriate size. The spatial forces acting at two points A and B on a rigid body are related as

$$F_B = P_{A,B} F_A$$

where $p_{A,B}$ denotes the position vector from B to .-1. If the linear and angular velocities of point A are zero then

$$\dot{V}_A = (\dot{P}_{A,B})^T \dot{V}_B$$

The matrix $P_{A,B}$ has the properties as $\dot{P}_{A,B} P_{B,C} = P_{A,C}$ and $(P_{A,B})^{-1} = P_{B,A}$.

The spatial inertia of link i about its center of mass, $I_{i,Ci}$, is given by

$$I_{i,Ci} = \begin{bmatrix} \mathrm{J}_i & 0 \\ 0 & m_i U \end{bmatrix} \varepsilon \Re^{6x6}$$

The spatial inertia of body i about point O_i (designated as I_i) is obtained as

$$I_i = \hat{S}_i I_{i,C_i} \hat{S}_i^T = \begin{bmatrix} U & \tilde{s}_i \\ 0 & U \end{bmatrix} \begin{bmatrix} \mathrm{J}_i & 0 \\ 0 & m_i U \end{bmatrix} \begin{bmatrix} U & 0 \\ -\tilde{s}_i & U \end{bmatrix}$$

$$= \begin{bmatrix} k_i & \tilde{h}_i \\ \tilde{h}_i^T & m_i U \end{bmatrix} \tag{2}$$

This represents the *parallel axis theorem* for propagation of spatial inertias.

In deriving the equations of motion, it is assumed that the nonlinear term $6(0, Q, F_E)$ is explicitly computed by using the recursive N-E algorithm. Similarly to the $O(N^3)$ and $O(N^2)$ algorithms [10,13], the explicit computation of $b(\theta, Q, F_E)$ provides additional parallelism in the present algorithm which can be exploited to further increase the speedup in the computation,

having computed the term $b(\theta, Q, F_E)$ and subsequently $\mathcal{F}_T$ in Eq. (1), the multibody system can be considered as a system at rest which, upon the application of control forces $\mathcal{F}_T$, accelerates in space. Accordingly, the linearized Newton-Euler equation of motion for rigid body i in the serial chain (Fig. 1) is given by

$$\dot{V}_i = P_{i-1}^T \dot{V}_{i-1} + H_i \ddot{Q}_i \tag{3}$$

$$F_i = I_i \dot{V}_i + P_i F_{i+1} \tag{4}$$

B. Interbody Force Decomposition Strategy

The iterative algorithms developed in [14,15] are based on a decomposition of interbody force of the form:

$$F_i = H_i F_{Ti} + W_i F_{Si} \tag{5}$$

where F_{Si} is the constraint force and W_i is the orthogonal complement of H_i [16,17], i.e.

$$W_i^T H_i = 0 \tag{6}$$

For joint i with multiple DOFs, say $k < 6$ DOFs, $H_i \varepsilon \Re^{6\times k}$ and $W_i \varepsilon \Re^{6\times(6-k)}$. Insofar as the axes of DOFs are orthogonal (which is the case considered in this paper) the matrix H_i is a projection matrix [16] arid hence

$$H_i^T H_i = U \tag{7}$$

It then follows that the matrix W_i is also a projection matrix, i.e.,

$$W_i^T W_i = U \tag{8}$$

An example of structure of matrices H_i and W_i for one-DOF revolute joint is given in §III. For a more detailed discussion on these matrices see [16,17].

The decomposition in Eq. (5) seems to be more natural (and perhaps more physically' intuitive) than that of the Articulated- Body Inertia (ABI) algorithm (Eq. (26) in [6]) since it expresses the interbody force in terms of two physical components: the control (or working) force and the constraint (or nonworking) force. That such a force decomposition has not been considered as a viable alternative for deriving algorithms for *direct* serial and parallel solution of the problem is not surprising. The decomposition in Eq. (5) naturally leads to the explicit computation of the constraint (and inter body) forces which has motivated the designation of the algorithm as constraint force algorithm. Indeed, researchers have often argued that since the constraint forces are nonworking forces, their explicit evaluation, which leads to the computational inefficiency, should be avoided. however, while this argument is in general valid for serial computation (which is also supported by our results), the explicit computation of the constraint forces, as shown below, results in highly efficient parallel algorithms for the problem.

C. A Schur Complement Factorization of $\mathcal{M}^{-1}$

In [1 1,12], we have shown that the force decomposition in Eq. (5) leads to a new factorization of $\mathcal{M}^{-1}$ as well as a new O(N) algorithm for the problem. Here, for the sake of completeness, this factorization of $\mathcal{M}^{-1}$ is briefly discussed.

First, let us rewrite Eqs. (3)-(4) as

$$\dot{V}_i - P_{i-1}^T \dot{V}_{i-1} = H_i \ddot{Q}_i \tag{9}$$

$$F_i - P_i F_{i+1} = I_i \dot{V}_i \tag{10}$$

and define a lower bidiagonal block matrix $\mathcal{P}$ as

$$\mathcal{P} = \begin{bmatrix} U & & & \\ -P_{N-1} & U & & \\ 0 & -P_{N-2} & U & \\ 0 & 0 & & \\ \vdots & \vdots & & \\ 0 & 0 & \cdots & U \end{bmatrix} \varepsilon \Re^{6N\times 6N}$$

Let us also define following global vector and matrix as

$$\mathcal{F}_S \triangleq \mathrm{col}\{F_{Si}\} \varepsilon \Re^{5N}$$

and

$$\mathcal{W} \triangleq \mathrm{diag}\{W_i\} \varepsilon \Re^{6N\times 5N} \quad i = N \text{ mboxto } 1$$

Equations (9)-(10) and (5)-(8) can now be written in global form as

$$\mathcal{P}^T \dot{\mathcal{V}} = \mathcal{H} \ddot{Q} \tag{11}$$

$$\mathcal{P}F = \mathcal{I}\dot{\mathcal{V}} \tag{12}$$

$$\mathcal{F} = \mathcal{H}F_T + \mathcal{W}F_S \tag{13}$$

$$\mathcal{H}^T\mathcal{W} = 0, \mathcal{W}^T\mathcal{H} = 0, \mathcal{H}^T\mathcal{H} = U, \text{and } \mathcal{W}^T\mathcal{W} = U \tag{14}$$

From Eqs. (11), (12), and (14) it follows that

$$\mathrm{v} = \mathcal{I}^{-1}\mathcal{P}F \tag{15}$$

$$\mathcal{W}^T\mathcal{P}^T\dot{\mathcal{V}} = \mathcal{W}^T\mathcal{H}\dot{\mathcal{Q}} = 0 \tag{16}$$

and from Eqs. (15)-(16), we get

$$\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}F = 0 \tag{17}$$

Substituting Eq. (13) into Eq.(17) yields

$$\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}(\mathcal{H}F_T + \mathcal{W}F_S) = \mathrm{o}$$

$$\Rightarrow \mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}W F_S = -\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}HF_T \tag{18}$$

or,

$$\mathcal{A}F_S = -\mathcal{B}F_T \tag{19}$$

where matrices $\mathcal{A} \triangleq \mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}W \varepsilon \Re^{5N\times 5N}$ and $\mathcal{B} \triangleq \mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}H \varepsilon \Re^{5N\times N}$ are block tridiagonal. By computing $\mathcal{F}_S$ from Eq.(18) and substituting it into Eq. (13), we get

$$\mathcal{F} = (\mathcal{H} - \mathcal{W}(\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}W)^{-1}\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}H)\mathcal{F}_T \tag{20}$$

and substituting Eq. (20) into Eq. (15) leads to

$$\dot{\mathcal{V}} = \mathcal{I}^{-1}\mathcal{P}\Big(\mathcal{H} - \mathcal{W}(\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}W)^{-1}\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}H\,\mathcal{F}_T\Big) \tag{21}$$

From Eqs. (11) and (14), $\dot{\mathcal{Q}}$ is computed as

$$\mathcal{H}^T\mathcal{H}\dot{\mathcal{Q}} = \mathcal{H}^T\mathcal{P}^T\dot{\mathcal{V}} \Rightarrow \mathrm{Q} = \mathcal{H}^T\mathcal{P}^T\dot{\mathcal{V}} \tag{22}$$

Finally, by replacing Eq. (21) into (22) it follows that

$$\mathrm{Q} = \Big(\mathcal{H}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}H - \mathcal{H}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}W(\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}W)^{-1}\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}H\,\mathcal{F}_T\Big) \tag{23}$$

which represents a compact operator form of the $O(N)$ CF algorithm. In comparison with Eq. (1), an operator form of $\mathcal{M}^{-1}$, in terms of its decomposition into a set of simple operators, is given as

$$\mathcal{M}^{-1} = \mathcal{H}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}H - \mathcal{H}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}W(\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}W)^{-1}\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}H \tag{24}$$

Letting $\mathcal{C} \triangleq \mathcal{H}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}H \varepsilon \Re^{N\times N}$, $\mathcal{M}^{-1}$ can now be expressed as

$$\mathcal{M}^{-1} = \mathcal{C} - \mathcal{B}^T\mathcal{A}^{-1}\mathcal{B} \tag{25}$$

where the matrix $\mathcal{C}$, similar to $\mathcal{A}$ and $\mathcal{B}$, is block tridiagonal. Furthermore, d and $\mathcal{C}$ are symmetric and positive definite (SPD). A proof of positive definiteness of $\mathcal{A}$ is given in [12] which guarantees the existence of $\mathcal{A}^{-1}$. A similar procedure can be used to prove the positive definiteness of $\mathcal{C}$.

The operator form of ***M-1 given*** by Eq. (25) represents an interesting mathematical construct. '1'0 see this, consider a matrix $\mathcal{L}$ defined as

$$\mathcal{L} \triangleq \begin{bmatrix} \mathcal{A} & \mathcal{B} \\ \mathcal{B}^T & \mathcal{C} \end{bmatrix} \varepsilon \Re^{6N\times 6N}$$

Then, $\mathcal{C} - \mathcal{B}^T\mathcal{A}^{-1}\mathcal{B}$ is the *Schur Complement* of d in C [18]. The structure of matrix $\mathcal{L}$ not only provides a deeper physical insight into the computation but it also motivates a different and somehow simpler approach for derivation of the algorithm [12].

III. An O(N) Serial Implementation of Constraint Force Algorithm

An efficient implementation of the CF algorithm is based on rewriting Eq. (23) as

$$\dot{\mathcal{Q}} = \mathcal{H}^T\mathcal{P}^T\Big(U - \mathcal{I}^{-1}\mathcal{P}W(\mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}W)^{-1}\mathcal{W}^T\mathcal{P}^T\Big)\mathcal{I}^{-1}\mathcal{P}HF_T \tag{26}$$

Here, the key to achieving greater efficiency in both serial and parallel computation is to simply perform, as much as possible, matrix-vector multiplication instead of matrix-matrix multiplication. In this regard, the matrices $\mathcal{B}$ and $\mathcal{C}$ do not need to be computed explicitly and only the explicit computation of matrix d is needed. Given $\mathcal{F}_T$, the computational steps of the algorithm then consist of a sequence of matrix-vector multiplications and a vector addition wherein the matrices, except for $\mathcal{A}^{-1}$, are either diagonal or bidiagonal. Multiplication of a vector by matrix d^{-1} is equivalent to the solution of a SPD block tridiagonal system, that is, the solution of

$$\mathcal{A}F_S = \dot{\mathcal{X}} \tag{27}$$

for $\mathcal{F}_S$ where $\dot{\mathcal{X}} \triangleq \mathrm{col}\{\dot{x}_i\} \varepsilon \Re^{5N}$, i = N to 1, and $\dot{\mathcal{X}} = \mathcal{B}F_T = \mathcal{W}^T\mathcal{P}^T\mathcal{I}^{-1}\mathcal{P}HF_T$.

Thus far, the solution procedure has been presented in a coordinate-free form. Before its implementation.

however, the tensors and vectors involved in the computation should be projected onto a suitable frame. The choice of the appropriate frame and the way t hat the projection is performed significantly affect the efficiency of the algorithm for both serial and parallel computation.

If the rotation of the one-DOF revolute joint i is given about the z axis of frame i then

$$H_i = \begin{bmatrix} z_i \\ 0 \end{bmatrix} \in \Re^6$$

The matrices H_i and W_i in frame i are given as

$$^{i}H_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad ^{i}W_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplication of any vector or matrix by $^{i}H_i$ and $^{i}W_i$ does not require any computation hut an appropriate permutation of the elements of the vector or matrix. However, in any other frame W_i is a dense matrix and its multiplication by another matrix requires a significant amount of computations. This clearly indicates that any projection of equations should be based on the maximum exploitation of the sparse structure of $^{i}W_i$ and $^{i}H_i$.

The matrix A and its elements are given as

$$A = \text{Tridiag}[B_i, A_i, B_{i-1}^T]$$

$$A_i = W_i^T(I_i^{-1} + \hat{P}_{i-1}^T I_{i-1}^{-1}\hat{P}_{i-1})W_i \quad i = N\text{to}1 \quad (28)$$

$$B_i = -W_i^T I_i^{-1}\hat{P}_i W_{i+1} \quad i = N-1\text{to}1 \quad (29)$$

in order to fully exploit the structure of $^{i}W_i$, A_i is computed in frame i as

$$A_i = {}^{i}W_i^T(^{i}I_i^{-1} + {}^{i}\hat{P}_{i-1}^T(^{i}I_{i-1})^{-1}{}^{i}\hat{P}_{i-1})^{i}W_i \quad (30)$$

By using the parallel axis theorem of Eq. (2), it can be shown that (Fig, 1)

$$\hat{P}_{i-1}^T I_{i-1}^{-1}\hat{P}_{i-1} = I_{i-1,i}^{-1} \quad (31)$$

That is, the term $\hat{P}_{i-1}^T I_{i-1}^{-1}$ Pi_1 is the inverse of the spatial inertia of link $i-1$ about point O_i which is constant in frame i and hence can be precomputed. (Note: according to our notation and Fig.1 frame i is fixed to link $i-1$ at its distal end.) It follows that the term $I_i^{-1} + \hat{P}_{i-1}^T I_{i-1}^{-1}\hat{P}_{i-1}$ in Eq. (30) is in fact the sum of inverse of spatial inertias of link i and link $i-1$ about their intersecting point O_i. The parallel axis theorem of Eq. (z) can be also used for propagat ion of inverse of spatial inertias as

$$I_i = S_i I_{i,C_i}\hat{S}_i^T \Rightarrow I_i^{-1} = (\hat{S}_i^T)^{-1} I_{i,C_i}^{-1}(\hat{S}_i^T)^{-1}$$

$$= \begin{bmatrix} J_i^{-1} & J_i^{-1}\tilde{s}_i \\ -\tilde{s}_i J_i^{-1} & -\tilde{s}_i J_i^{-1}\tilde{s}_i + (1/m_i)U \end{bmatrix} \quad (32)$$

Both I_i and I_i^{-1} are constant in frame $i+1$, However, while 1, has a simple and sparse structure, I_i^{-1} has a dense structure. It is more efficient to first project J_i^{-1} and s_i onto frame i as

$$^{i}J_i = c(i,i+1)\,{}^{i+1}J_i c(i+1,i) \Rightarrow$$

$$(^{i}J_i)^{-1} = c(i,i+1)\,(^{i+1}J_i)^{-1}c(i+1,i) \quad (33)$$

$$^{i}s_i = c(i,i+1)\,{}^{i+1}s_i \quad (34)$$

and then compute $(^{i}I_i)^{-1}$ in frame i according to Eq. (32). The computation of B_i is also performed in frame i as follows. Let us define

$$\Psi_i \triangleq I_i^{-1}\hat{P}_i W_{i+1} \Rightarrow {}^{i+1}\Psi_i = (^{i+1}I_i)^{-1}\,{}^{i+1}\hat{P}_i\,{}^{i+1}W_{i+1}$$

'I'he matrix Ψ_i is constant in frame $i+1$ and can be precomputed. It is projected onto frame i as

$$^{i}\Psi_i = C(i,i+1)\,{}^{i+1}\Psi_i \quad (35)$$

Then B_i can be computed as

$$B_i = -{}^{i}W_i^T\,{}^{i}\Psi_i \quad (36)$$

which does not need any computation but a permutation of matrix $^{i}\Psi_i$. To exploit further the sparse structure of $^{i}W_i$ and $^{i}H_i$, the rest of the computations in Eq. (26) is also projected onto frame i.

The block tridiagonal system can be solved by both block odd-even cyclic reduction algorithm [19-20] and block LDL^T factorization [31] in O(N) steps. However, for serial computation, the latter algorithm is more efficient by a factor of ≈ 2.5 [19].

The computation of the serial CF algorithm is performed as follows:

Step I. Compute $\mathcal{F}_T$ by using the N-E algorithm in *[4]* with $\dot{Q} = O$.

Step II. Compute $\ddot{\mathcal{X}} = \text{c o1}\ \{\ddot{x}_i\}$

A. Projection

1. Form $c(i,i+1)$ and $C(i,i-t\ 1)$.
2. Form $^{i}z_{i+1}$ = third column of $c(i, i+1)$.
3. Compute $(^{i}J_i)^{-1}$ and $^{i}s_i$ from Eqs. (33)-(34).
4. Compute $(^{i}I_i)^{-1}$ from Eq. (32).

B. Compute $\dot{\mathcal{V}}^1$

1. Compute $\mathcal{F}_T^1 \triangleq \text{col}\{F_{Ti}^1\} = \mathcal{H} F_T \varepsilon \Re^{6N}$

$$^iF_{Ti}^1 = {}^iH_i F_T \qquad (37)$$

2. Compute $\mathcal{F}_T^2 \triangleq \text{col}\{^iF_{Ti}^2\} \varepsilon \Re^{6N}$

$$^iF_{Ti}^2 = {}^iF_{Ti}^1 - C(i, i+1)^{i+1}\dot{P}_i {}^{i+1}F_{Ti+1}^1 \qquad (38$$

3. Compute $\dot{\mathcal{V}}^1 \triangleq \text{col}\{^i\dot{V}_i^1\} = \mathcal{I}^{-1}\mathcal{F}_T^2 \varepsilon \Re^{6N}$

$$^i\dot{V}_i^1 = (^iI_i {}^{-1i}F_{Ti}^2 \qquad (39$$

C. Compute $\dot{\mathcal{X}} = \mathcal{W}^T\mathcal{P}^T\dot{\mathcal{V}}^1$

1. Compute $\dot{\mathcal{V}}^2 \triangleq \text{col}\{^i\dot{V}_i^2 = \mathcal{P}^T\dot{\mathcal{V}}^1$

$$^i\dot{V}_i^2 = {}^i\dot{V}_i^1 - {}^i\dot{P}_{i-1}^T C(i-1)^{i-1}\dot{V}_{i-1}^1 \qquad (40)$$

2. Compute $\dot{\mathcal{X}} = \mathcal{W}^T\dot{\mathcal{V}}^2$

$$\dot{x}_i = {}^i W_i^{Ti}\dot{V}_i^2 \qquad (41)$$

Step 111. Form matrix $\mathcal{A}$

A. Compute B_i, i = $N-1$ to 1, from Eqs. (35)-(36).

B. Compute A_i, i = N to 1, from Eq. (30).

Step IV. Solve $\mathcal{A}F_S = \dot{\mathcal{X}}$ for $\mathcal{F}_S \varepsilon \Re^{5N}$ by using the block LDL^T algorithm

Step V. Compute $\dot{\mathcal{Q}}$

A Compute $\dot{\mathcal{V}}^3 = \mathcal{I}^{-1}\mathcal{P}\mathcal{W}F_S$

1. Compute $\mathcal{F}_S^1 \triangleq \text{col}\{^iF_{Si}^1\} = \mathcal{W}F_S \varepsilon \Re^{6N}$

$$^iF_{Si}^1 = {}^iW_i^1 F_{Si} \qquad (4'2)$$

2. Compute $\mathcal{F}_S^2 \triangleq \text{col}\{^iF_{Si}^2\} = \mathcal{P}F_S^1 \varepsilon \Re^{6N}$

$$^iF_{Si}^2 = {}^iF_{Si}^1 - C(i, i+1)(^{i+1}\dot{P}_i^{i+1}F_{Si+1}^1) \qquad (43)$$

3. Compute $\dot{\mathcal{V}}^3 \triangleq \text{col}\{^i\dot{V}_i^3\} = \mathcal{I}^{-1}\mathcal{F}_S^2 \varepsilon \Re^{6N}$

$$^i\dot{V}_i^3 = \textit{(iii)-liF~i} \qquad (44)$$

B. Compute $\dot{\mathcal{Q}} = \mathcal{H}^T\mathcal{P}^T(\dot{\mathcal{V}}^1 - \dot{\mathcal{V}}^3)$

1. Compute $\dot{\mathcal{V}}^4 \triangleq \text{col}\{^i\dot{V}_i^4\} = (\dot{\mathcal{V}}^1 - \dot{\mathcal{V}}^3) \varepsilon \Re^{6N}$

$$^i\dot{V}_i^4 = {}^i\dot{V}_i^1 - {}^i\dot{V}_i^3 \qquad (45)$$

2. Compute $\dot{\mathcal{V}}^5 \triangleq \text{col}\{^i\dot{V}_i^5\} = \mathcal{P}^T\dot{\mathcal{V}}^4 \varepsilon \Re^{6N}$

$$^i\dot{V}_i^5 = {}^i\dot{V}_i^4 - {}^i\dot{P}_{i-1}^T C(i, i-1)^{i-1}\dot{V}_{i-1}^4 \qquad (46)$$

3. Compute $\dot{\mathcal{Q}} = \mathcal{H}^T\dot{\mathcal{V}}^5$

$$\dot{Q}_i = {}^iH_i^{Ti}\dot{V}_i^5 \qquad (47)$$

The proposed scheme for serial implementation of the CF algorithm seems to be optimal and, in fact, it is unlikely that any further significant improvement in efficiency of the algorithm can be achieved. However, despite this optimal implementation, the CF algorithm is less efficient than other algorithms for serial computation. Note that Step I is common to both the CF algorithm and the ABI algorithm. Excluding its cost, the computational cost of the CF algorithm is (695m+592a)N – (408m+347a) while the cost of ABI algorithm, as reported in [6], is, (199m+174a)N – (198m+ 173a). Hence, for large N, the ABI algorithm is more efficient than the CF algorithm for serial implementation by a factor of $\approx$ 3.4 (ill terms of total number of operations). Obviously, for small N (say N < 12), the CF algorithm is also less efficient than the $O(N^2)$ or $O(N^3)$ algorithms.

IV. Parallel Computation of Constraint Force Algorithm

A. Parallelism in the CF Algorithm: Time and Processor Bound in Computation

The efficiency of the CF algorithm for parallel computation can be easily assessed by examining the steps involved in its serial implementation. By using the parallel algorithm of [22], Step I can be performed in $O(Log N)$ -t O(1) with O(N) processors. Steps 11, 111 and V are completely parallelizable since the computations for each body are decoupled. Hence, with 0(N) processors, these steps can be performed with a computational cost of $O(1)$. The main issue in parallelization of the CF algorithm is then the parallel solution of block tridiagonal system in Step IV.

As discussed before, the block LDL^T factorization is the most efficient serial algorithm for solution of block tridiagonal systems. However, this algorithm seems to be strictly serial and, in fact, there is no report on its parallelization. On the other hand, the block cyclic reduction algorithm, while not the most efficient for serial computation, can be parallelized and performed in $O(Log N)$ + 0(1) steps with O(N) processors [20].

Therefore, it can be concluded that the entire CF algorithm. from Step I to Step V can be performed in

$O(LogN) + O(1)$ with $O(N)$ processors, This represents a both time and processor-opt imal parallel algorithm for the problem.

B. Parallel Solution of SPD Block Tridiagonal Matrix

A central issue that affects the parallel efficiency of the CF algorithm is the choice of parallel algorithm for solution of block tridiagonal system. There are two variants of the cyclic reduction algorithm: the Odd-Even Reduction (OER) and Odd-Even Elimination (OEE) algorithms [20]. The OEE algorithm, while less efficient than the OER algorithm for serial computation, provides additional parallelism (by about a factor two) in both computation and communication using the same number of processors and interconnection structure as for OER algorithm [20]. For solution of the block tridiagonal system in Eq. (27) the OEE algorithm is given as follows:

For $i = N$ to 1, Do

$A_i^0 = A_i$, $B_i^0 = B_i$, and $\dot{x}_i^0 = \dot{r}_i$ (initialization)

End Do

For $j = 1$ to $M = \lceil Log_2 N \rceil$, Do

For i $= N$ to 1, Do

$$A_i^j = A_i^{j-1} - B_i^{j-1}(A_{i+2^{j-1}}^{j-1})^{-1}(B_i^{j-1})^T - (B_{i-2^{j-1}}^{j-1})^T(A_{i-2^{j-1}}^{j-1})^{-1}B_{i-2^{j-1}}^{j-1} \tag{48}$$

$$B_i^j = -B_i^{j-1}(A_{i+2^{j-1}}^{j-1})^{-1}B_{i+2^{j-1}}^{j-1} \tag{49}$$

$$\dot{x}_i^j = \dot{x}_i^{j-1} - B_i^{j-1}(A_{i+2^{j-1}}^{j-1})^{-1}\dot{x}_{i+2^{j-1}}^{j-1} - (B_{i-2^{j-1}}^{j-1})^T(A_{i-2^{j-1}}^{j-1})^{-1}\dot{x}_{i-2^{j-1}}^{j-1} \tag{50}$$

End Do

End [)0

For $i = N$ to 1, Do

$$Solve\ A_i^M F_{si} = \dot{x}_i^M \text{ for } F_{si} \tag{.51}$$

End Do

where $\lceil x \rceil$ indicates the smallest integer greater than or equal to x. It should be noted that in Eqs. (48)-(49) it is more efficient to first compute the scalar ldl^T factorization of the dense submatrices rather than their explicit inverses. The multiplication of the inverse of a matrix by another matrix, e.g., $(A_{i+2^{j-1}}^{j-1})^{-1}(B_i^{j-1})^T$ in Eq.(50), can then be computed as the solution of a linear system with multiple right-hand sides.

The parallel implementation of both the OER and OEE algorithms for scalar tridiagonal systems is straightforward. However, for block tridiagonal systems care should be taken to achieve the optimal efficiency. In fact, it seems that efficient implementation of eit her algorithm for block tridiagonal systems has received less attention in the literature. Note that an implementation strategy starts with a specific process-to-processor allocation scheme. To see this, consider the parallel implementation by using N processors, designated as PR_i, $i = N$ to 1, Let us further assume a perfect mapping of the algorithm, i.e., a mapping on an architecture with N processors and a Shuffle-Exchange augmented with Nearest Neighbor (SENN) interconnection structure, There are two possible strategies for parallel implementation of the OEE algorithm given above. In the first and more obvious strategy the computation of A_i^j, B_i^j, $\dot{x}_i^j$, and Fs_i as well as all the intermediate terms in Eqs. (48) – (51) is assigned to processor PR_i. Note that this strategy, which seems to be widely adopted in the literature for implementation of both the OER and OEE algorithms, is optimal for scalar tridiagonal systems,

We have developed a second strategy in which the terms A_i^j and $\dot{x}_i^j$ as well as all intermediate terms involving A_i^{j-1} are computed by PR_i. These two strategies lead to two different structures for the computat ion performed by each processor as well as the communication among processors, The impact of the two strategies on both computation and communicaon complexity of the algorithm is discussed in [23]. It will suffice to mention here that the second strategy, presented below, is more efficient for solution of block tridiagonal systems since it leads to a slightly greater computational efficiency. More importantly, it also provides a high degree of overlapping between the computation and the communication which can be exploited to reduce the communication overhead.

Using our strategy, the parallel implementation of the OEE algorithm is given by:

For j = 1 to $M = \lceil Log_2 N \rceil$, Do

For $i = 1$ to N, Do Parallel (by all PR_i 's)

1. Compute ldl^T factorization of A_i^{j-1}

(52)

2. Solve $A_i^{j-1} C_i^{j-1} = B_i^{j-1}$ for C_i^{j-1}

(53)

3. Compute $D_i^{j-1} = (B_i^{j-1})^T C_i^{j-1}$

(54)

4. Compute $E_i^{j-1} = (B_i^{j-1})^T (A_i^{j-1})^{-1} \dot{x}_i^{j-1} = (C_i^{j-1})^T \dot{x}_i^{j-1}$

(55)

5. Send D_i^{j-1} and E_i^{j-1} to $PR_{i+2^{j-1}}$

6. Solve $A_i^{j-1} F_i^{j-1} = (B_{i-2^{j-1}}^{j-1})^T$ for F_i^{j-1}

(56)

7. Compute $G_i^{j-1} = (B_{i-2^{j-1}}^{j-1})^T (A_i^{j-1})^{-1} (B_{i-2^{j-1}}^{j-1})^T = (B_{i-2^{j-1}}^{j-1})^T F_i^{j-1}$

(57)

8. Compute $H_i^{j-1} =$ Ii:::;, -,(.A; -')-lij-l $= (F_i^{j-1})^T \dot{x}_i^{j-1}$

(58)

9. Send G_i^{j-1} and H_i^{j-1} to $PR_{i-2^{j-1}}$

10. compute $(B_{i-2^{j-1}}^{j})^T = (B_i^{j-1})^T (A_i^{j-1})^{-1} (B_{i-2^{j-1}}^{j-1})^T = (B_i^{j-1})^T F_I^{j-1}$

(59)

11. Send $(B_{i-2^{j-1}}^{j})^T$ to $PR_{i-2^{j-1}}$

12. Compute $A_i^j = A_i^{j-1} - D_{i-2^{j-1}}^{j-1} - G_{i+2^{j-1}}^{j-1}$

(60

13. compute $\dot{x}_i^j = \dot{x}_i^{j-1} - E_{i-2^{j-1}}^{j-1} - H_{i+2^{j-1}}^{j-1}$

(61)

14. Send $(B_{i-2^{j-1}}^{j})^T$ to $PR_{i+2^{j-1}}$

(62)

End Do Parallel

End Do

For r' = 1 to N, Do Parallel (by all PR_i's)

1. Compute $/d/^T$ factorization of A_i^M

(63

2. Solve $A_i^M F_{si} = \dot{x}^M$ for F_{si}

(64

End 1)0 Parallel

As can be seen, any communication activity of processor PR_i can be overlapped by its immediate computation activity. That is, communication of D_i^{j-1} and E_i^{j-1} can be overlapped with computation of F_i^{j-1}; communication of G_i^{j-1} and H_i^{j-1} can be overlapped with computation of $(B_{i-2}^{j} j - 1)7'$: and finally communication of $(B_{i-2}^{j} j-1)^T$ can be overlapped with computation of A_i^j and $\dot{x}_i^j$. This overlapping feature is particularly suitable for implementation on MIMD architectures such as the Hypercube since it can be exploited to significantly reduce the communication overhead.

C. Performance of Perfect Mapping of' the Parallel CF Algorithm

in [1 1,12], we analyzed the performance of the parallel CF algorithm by considering a perfect mapping of the algorithm, i.e., its implementation on an architecture with N processors and with a SENN interconnection. This analysis indicated that such an implementation will result in a computation cost of $(732m+653a)Log_2 N + (542m+439a)$ and a communication cost of $(10\beta+134\alpha)LogN_2+(6\beta+49\alpha)$, where β and α stand for the cost (time) of communication start-up and the cost of communicating a single datum, respectively. Note that in analyzing the communication cost the overlapping of communication and computation is not considered,

Such a theoretical performance clearly indicates that the CF algorithm, though not efficient for serial computation, has excellent features for parallel computation. The parallel CF algorithm not only achieves the theoretical time lower bound of $O(LogN)$ + 0(1) but is also highly practical from an implementation perspective. First, Steps II, 111, and IV- which have a serial computation complexity of O(N)- can be fully parallelized and performed in $O(1)$ with limited nearest neighbor communication among processors.

Second, the parallel algorithm is highly compute bound, i.e., its communication cost is much smaller than its computation cost. Note that the coefficient of β indicates the level of communication activities which, as can be seen, is very low. Also, as stated before, our strategy for implementing the OEE algorithm is specifically motivated by the overlapping capability which can be exploited to further reduce the communication overhead. Coupled with the limted communication activities is the fact that the algorithm has a rather coarse grain size since, particularly in Steps II-V, each processor performs a matrix-vector operation or a sequence of such operations before communicating to other processors. The coarse grain feature and limited communication requirements make the CF algorithm highly suitable for implementation on MIMD parallel architectures such as the Hypercube.

D. Strategies for Multilevel Parallel Implementation of the CF Algorithm on a Hypercube

The parallel implementation of the CF algorithm with N processors achieves an asymptotically optimal parallel computational complexity and a significant speedup for large N. However, for small N, it offers a rather limited speedup [1 1,12], which is due to the large coefficient of the $Log_2 N$ terms. Note that this phenomenon represents an inherent algorithmic property in both serial and parallel computation of the multibody dynamics problem. That serial O(N) algorithms, though asymptotically optimal, are less efficient than the serial $O(N^2)$ or $O(N^3)$ algorithms for small N is also due to the large coefficient of N.

"1o increase the efficiency of the parallel CF algorithm for small N, the coefficient of the $Log_2 N$ term needs to be reduced. This can be achieved by further exploiting parallelism through a multilevel approach, A first possible strategy is to exploit fine-grain parallelism in various matrix-vector operations of the algorithm. However, this strategy would require the implementation of the algorithm on special-purpose parallel architectures such as the one proposed in [24].

here, an alternative coarse-grain multilevel parallel computation approach is presented which is particularly suitable for implementation on the Hypercube. This approach is moreover motivated by the fact that, for small N, it is likely that the number of processors of the target architecture be much greater than N. This naturally suggests the possibility of increasing the efficiency of the algorithm by using more of the processors that would otherwise be idle.

Step IV represents the most computation intensive part of the parallel algorithm. Therefore, any effective multilevel approach should be based on exploitation of maximum parallelism in computation of this step. However, as shown below, further efficiency can be also achieved by modifying the computation of this step so it can be overlapped with other steps of the parallel algorithm.

The OEE algorithm given by Eqs. (48)–(51) can be interpreted as a procedure **for diagonalization of block** tridiagonal **matrix $\mathcal{A}$ in which a** series of transformations are applied to both sides of Eq. (27), resulting in a block diagonal system given by ***Eq. (51).*** That is, Eqs. (48)-(49)—or **Eqs.** (52)-(54), (56)-(57), and (59)-(60) of our variant of OEE algorithm- -represent the diagonalization of matrix $\mathcal{A}$ while Eq. (50)- or Eqs. (55), (58), and (61)-–represents the updating of the right-hand side. Taking this perspective, the computation of the OEE algorithm can be broken into two parts: the diagonalization of matrix $\mathcal{A}$, i.e., computation of $\mathcal{A}^M = \text{col}\{A_i^M\}$, in which the submatrices A_i^j and B_i^j are computed; and the updating of the right-hand side, i.e., computation of $\dot{\mathcal{X}}^M = \text{col}\{\dot{x}_i^M\}$, by using the already computed submatrices A_i^j and B_i^j. The diagonalization of matrix $\mathcal{A}$ and the computation of $\mathcal{F}_T$ and $\dot{\mathcal{X}}$, i.e., Steps I and II of previous section, arc fully decoupled and can be performed in parallel in an overlapped and completely asynchronous fashion.

E. A Two-Level Parallel Computation Strategy

Our first multilevel approach, which represents a two-level parallel computation, is based on the above decomposition of the CF algorithm. In this approach, the CF algorithm is implemented on two groups of processors wherein each group consists of N processors. On the Hypercube, processors numbered O to N – 1 form the first group and processors numbered N to 2N – 1 form the second group. The computation of matrix A and its reduction are assigned to the first group while the the computation of $\mathcal{F}_T$ and $\dot{\mathcal{X}}$ are assigned to the second group. Since these two sets of computation are completely decoupled, the activities of the two groups are carried in parallel and in fully asynchronous fashion.

Our timing results indicate that the computation and reduction of matrix $\mathcal{A}$ take more time than the computation of $\mathcal{F}_T$ and $\dot{\mathcal{X}}$. Therefore, the overall computation time is dominated by that of computation and reduction of matrix $\mathcal{A}$ and the computation of $\mathcal{F}_T$ and $\dot{\mathcal{X}}$ is fully overlapped, This also implies that, upon computation of $\dot{\mathcal{X}}$, the processors of the second group have to wait for the completion of the reduction of matrix $\mathcal{A}$ by the processors of the first group,

In order to further increase, the computational efficiency, the computation of $\dot{x}_i^j$, i.e., Eq.(50) or Eqs. (55), (58), and (61) of our variant of the OEE algorithm, is also assigned to the second group of processors. To do so, the processors of the first group, after computing $(C_i^{j-1})^T$ and $(F_i^{j-1})^T$, send them to the processors of the second group. Note that this communication activity is performed asynchronously and, furthermore, it can also be overlapped with the computation. With this scheme, the computation of $\dot{x}_i^j$ is also mostly overlapped with the reduction of matrix A. In this two-level strategy, the computation of Eqs. (62)-(63) as well as the computation of Step V of the CF algorithm, i.e., Eqs. (42)-(47), are also assigned to the second group of processors.

F. A Three-Level Parallel Computation Strategy

In the two-level parallel implementation of the CF algorithm, the reduction of matrix $\mathcal{A}$ remains the most time-consuming part which also dominates the overall computation time. This has motivated us to develop a second multilevel strategy, or a three-level parallel

computation, to further speed up the computation by exploiting a higher degree of parallelism in the reduction of matrix $\mathcal{A}$.

Algorithm	Time (in ms)	Speedup Absolute	Speedup Relative
Serial	481	1.00	-
N-Parallel	87	5.53	5.53
2N-Parallel	69	6.97	1.26
3N-Parallel	55	8.75	1.25

Table I. Computation Time of Serial and Parallel Multibody Implementation of the CF Algorithm

In this strategy the parallel computation of the CF algorithm is performed by using three groups of processors. On the Hypercube, processors number O to N – 1 form the first group, processors number N to 2N – 1 form the second group, and processors number 2N to 3N – 1 form the third group, The computation of matrix d is assigned to the first group while the reduction of matrix $\mathcal{A}$ is performed by both first and second groups. This is achieved by decomposing the computation of Eq. (48) and by using processor PR_i, $i < N$, of the first group and processor PR_{i+N} of the second group for computation of A_i^I Note that if N is a power of 2 (which is the case considered here) then processors PR_i, i < N, and processor PR_{i+N} are physically nearest neighbors (i. e., there is a direct link between them), The computation of $\mathcal{F}_T, \dot{\mathcal{X}}$, Eqs. (62)-(63), and Step V of the CF algorithm, are assigned to the third group.

V. Discussion and Conclusion

We have implemented the parallel CF algorithms on the Intel iPSC/2 Hypercube installed at UC Irvine, This is a 32-node Hypercube with each node having an 80386 microprocessor, an 80387 arithmetic coprocessor, and 4 MBytes RAM. **This number of nodes limited** our implementation to the case of a multibody system with 8 DOF (N = 8),

The computation time of serial and multilevel parallel CF algorithms are shown in Table I. In order to evaluate the performance of the parallel CF algorithms, we have implemented the serial algorithm of §III on one node of the Hypercube. As can be seen, the N-Parallel implementation of the CF algorithm on such MIMD architecture results in a rather significant speed up. This clearly indicates that the CF algorithm is highly efficient for practical implementation. Our two- and three-level parallel computation strategies also improve the efficiency of parallel computation. however, it should be emphasized that, for large N, the parallel CF algorithm wit] result in even greater speedup in the computation. Currently, we are implementing the CF algorithm for larger problems (N = 16) on JPL's Hypercube.

Acknowledgments

The research of the first author was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration (NASA),

REFERENCES

1. A. Fijany, "Parallel $O(LogN)$ Algorithms for Open- and Closed-Chain Rigid Multibody Systems Based on a New Mass Matrix Factorization Technique," Proc. 5th NASA Workshop on Aerospace Computational Control, Aug. 1992.

2. A. Fijany, "New Factorization Techniques and Parallel $O(LogN)$ algorithms for Computation of Single Closed-Chain Robot Manipulators Forward Dynamics," Submitted to IEEE Trans. Syst., Man, and Cybern.. Also, in JPI, Engineering Memo, EM 343-1315-93, July 1993,

3. J.Y.S. Luh, M.W. Walker, and R.P.C. Paul, "On-Line Computational Scheme for Mechanical Manipulator," ASME J. Dynamic Syst., Meas., Control, Vol. 102, pp. 69-76, June 1980.

4. M.W. Walker and D.E. Orin, "Efficient Dynamic Computer Simulation of Robotic Mechanism, " ASME J. Dynamic Systems, Measurement, and Control, Vol. 104, pp. 205-211, 1982,

5. D.E. Rosenthal, "Triangularization of Equations of Motion for Robotic Systems, " J. Guidance, Control, and Dynamics Vol. 11, pp. 278-281, 1988.

6. R. Featherstone, "The Calculation of Robot Dynamics Using Articulated- Body Inertia," Int. J. Robotics Research, Vol. 2(l), pp. 13-30, 1983.

7. G. Rodriguez, "Kalman Filtering, Smoothing and Recursive Robot Arm Forward and Inverse Dynamics," IEEE J. Robotics and Automation, Vol. RA-3(6), pp. 624-639, Dec. 1987.

8. G. Rodriguez and K. Kreutz-Delgado, "Spatial Operator Factorization and Inversion of the Manipulator Mass Matrix," IEEE Trans. Robotics and Automation, Vol. RA-8(1), pp. 65-76, Feb. 1992.

9. D. Rosenthal, "Order N Formulation for Equations of Motion of Multi body Systems, " Proc. SDIO/NASA Workshop on Multibody Simulation, pp. 1122-1150, Sept. 1987.

10. A. Fijany and A.K. Bejczy, "Techniques for Parallel Computation of Mechanical Manipulator Dynamics.Part II: Forward Dynamics, " in *Advances in Control and Dynamic Systems, Vol.* 40: *Advances in Robotic Systems Dynamics and Control* C.T. Leondes (Ed,), pp. 357-410, Academic Press, March 1991.

11. A. Fijany, I. Sharf, and G. M.T. D'Eleuterio, "Parallel $O(LogN)$ Algorithms for Computation of Manipulator Forward Dynamics, " Submitted to IEEE Trans. Robotics and Automation,

12. A. Fijany, "Parallel $O(LogN)$ Algorithms for Rigid Multibody Dynamics, " JPL Engineering Memo, EM 343-W-1258, Aug. 1992,

13. A. Fijany and R.E. Scheid, "Fast Parallel preconditioned Conjugate Gradient Algorithms for Computation of Manipulator Forward Dynamics," To appear in J. of Intelligent & Robotic Systems: Theory & Application, 1993. Also, in JPL Eng. Memo, EM 343-91-1196, Aug. 1991,

14. I. Sharf, ***Parallel*** *Simulation Dynamics for* ***Open*** *Multibody Chains*, Ph.D. Diss., Univ. of Toronto, Canada, Nov. 1990.

15. I. Sharf and G.M.T. 1) 'Eleuterio, "Parallel Simulation Dynamics for Rigid Multibody Chains, " Proc. 12th Biennial ASME Conf. on Mechanical Vibration and Noise, Montreal, Canada, Sept. 1989.

16. P.C. Hughes and G.B. Sincarsin, "Dynamics of an Elastic Multibody Chain: Part B—Global Dynamics," Dynamics and Stability of Systems, Vol. 4(3&4), pp. 227-244, 1989,

17 C.J. Damaren and G.M.T. D'Eleuterio, "On the Relationship between Discrete-Time Optimal Control and Recursive Dynamics for Elastic Multibody Chains, " Contemporary Mathematics, Vol. 97, pp. 61-77, 1989.

18. R.W. Cottle, "Manifestation of Schur Complement," Linear Algebra and its Application, Vol. 8, pp. 189-211, 1974.

19. D. Heller, "Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems," SIAM J. Numer. Anal., Vol. 13(4), 1976.

20. R.W. Hockney and C.R. Jesshope, *Parallel Computers*, Adam Hilger Ltd, 1981.

21. G, II, Golub and C.F. Van Loan, *Matrix Computations* 2nd Edition, The John Hopkins Univ. Press, 1989.

22. C.S.G. Lee and P.R. Chang, "Efficient Parallel Algorithms for Robot Inverse Dynamics Computation ," IEEE Trans. Syst., Man, and Cybern., Vol. 16(4), pp. 532-542, July/August 1986.

23. A. Fijany, N. Bagherzadeh, and G. Kwan, "Communication Efficient Cyclic Reduction Algorithms for Parallel Solution of Block Tridiagonal System, " Submitted to Information Processing Letters.

24. A. Fijany and A.K. Bejczy, "ASPARC: An Algorithmically Specialized Parallel Architecture for Robotics Computations," in ***Parallel*** *Computation* ***Systems for Robotics: Algorithms and*** *Architectures*, A. Fijany and A.K. Bejczy (Eds.), World Scientific Pub., 1992.